ELSEVIER

Theoretical Computer Science 146 (1995) 185–197

Theoretical Computer Science

Formalizing Dijkstra's predicate transformer wp in weak second-order logic

Rudolf Berghammer[a,*], Birgit Elbl[b], Ulf Schmerl[b]

[a] *Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, Preusserstr. 1-9, 24105 Kiel, Germany*

[b] *Fakultät für Informatik, Universität der Bundeswehr München, Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany*

Received June 1992; revised February 1994
Communicated by M. Sintzoff

Abstract

We present a purely syntactical but nevertheless handy definition of Dijkstra's predicate transformer wp in weak second-order logic. This formalization allows us to prove a normal form theorem of wp, from which interesting properties can be inferred. In particular, it turns out that $\mathrm{wp}(P,\cdot)$ is a homomorphism on the class of formulae in the formalism considered.

1. Introduction

Based on the axiomatic method of [8], in [3,4] an approach to reasoning about total correctness of imperative programs has been proposed, which is based on the concept of weakest preconditions. This approach has been carried further by a number of people. In particular, in [1] it is shown how Dijkstra's definition of the predicate transformer $w\wp$ on sets of states rigorously can be formalized in the infinitary first-order logic $\mathcal{L}_{\omega_1\omega}$ (see e.g. [9, 10]) as a function mapping a program of Dijkstra's nondeterministic programming language of guarded commands and an infinite formula to an infinite formula. In the same article it is also proved that the weakest preconditions cannot be expressed in the usual finitary first-order logic if the underlying programming language contains loops. A formalization in higher-order logic is considered in [2].

The first objective here is also to present a rigorous syntactical formalization of weakest preconditions. In contrast to [1], however, we do not use infinitely long

* Corresponding author. E-mail: rub@informatik.uni-kiel.d400.de

SSDI 0304-3975(94)00184-7

formulae but formalize Dijkstra's predicate transformer as a function on finite formulae of a language of weak second-order logic. That this logic is useful for reasoning about programs had been pointed out before for instance in [12]. Our second objective is to examine the basic properties of predicate transformers (see e.g. [4], Chs. 4 and 9]). When considering a formalization of $w\wp$ and a corresponding deductive system, the question arises whether these properties are already *derivable*. Of course the answer is "yes" as soon as a complete deductive system is used, but this is unsatisfactory for logics like $\mathscr{L}_{\omega_1\omega}$, because here complete deductive systems necessarily make use of infinite derivations, i.e., they are only semi-formal systems. We show that these conditions can be derived without infinitary reasoning. In particular, we prove a "normal form" property of wp. From this normal form it easily follows that $\mathrm{wp}(P,\cdot)$, for a fixed program P, can be described as a monotonic homomorphism on the class of weak second-order formulae. We are indebted to the anonymous referee of a preliminary version of this paper for having called our attention to this point. For a treatment of properties of predicate transformers in another formalism see e.g. [5]. The formalization we were looking for has to be finite – as already mentioned – but also *practicable*: the weakest preconditions are used as a framework in which to establish the soundness of verification techniques. A practicable formalization should be well suited for this purpose. To meet this postulate can be regarded as our third objective.

2. A simple programming language and the predicate transformer $w\wp$

Let $\Sigma = (S, C, F)$ be a signature, where S, C, F are the set of sorts, constant symbols and function symbols, respectively. S is supposed to contain bool and nat. For every $s \in S$, PV_s denotes a set of names used as the programming variables of sort s and $\mathrm{PV} := \bigcup_{s\in S} \mathrm{PV}_s$. The set EXP_s of expressions of sort $s \in S$ over Σ and PV is defined as usual. Concerning C, F, PV as well as the sets of symbols to be defined later, we postulate that syntactical objects of different sorts have different names. Now we define the set of L_p-statements.

Definition 2.1. The set of statements STAT is inductively defined as follows:

- **skip**, **abort**, and the assignments $(x := t)$, where $x \in \mathrm{PV}_s$ and $t \in \mathrm{EXP}_s$, are elements of STAT.
- If $P_1, P_2 \in \mathrm{STAT}$ and $b \in \mathrm{EXP}_{\mathrm{bool}}$, then

  $(P_1; P_2)$, **if** b **then** P_1 **else** P_2 **fi**, and **while** b **do** P_1 **od**

  are elements of STAT.

Later on we will also discuss a nondeterministic variant of L_P. $\mathrm{STAT}^{\mathrm{nondet}}$ denotes the set of statements of this variant and is defined similar to STAT except for the

conditional statement. This is replaced by

if b_1 **then** P_1 ▯ b_2 **then** P_2 ▯ ⋯ ▯ b_n **then** P_n **fi**,

where $b_1, b_2, \ldots, b_n \in \mathrm{EXP}_{\mathrm{bool}}$ and $P_1, P_2, \ldots, P_n \in \mathrm{STAT}^{\mathrm{nondet}}$.

A fixed Σ-algebra $\mathscr{A} = ((s^{\mathscr{A}})_{s\in S}, (c^{\mathscr{A}})_{c\in C}, (f^{\mathscr{A}})_{f\in F})$ is used to determine the meaning of the symbols of the signature Σ. The sorts $s \in S$ are interpreted as sets, the function symbols $f \in F$ as total functions. We choose the standard interpretation for the sorts bool and nat and for the symbols which denote the usual operations on them. STATE is the set of value assignments in $\mathscr{A}$ (respecting the sorts, of course), i.e., the set of mappings $v: \mathrm{PV} \to \bigcup_{s\in S} s^{\mathscr{A}}$ which meet the condition $x \in \mathrm{PV}_s$ iff $v(x) \in s^{\mathscr{A}}$. Given an expression $t \in \mathrm{EXP}_s$ and an assignment $v \in \mathrm{STATE}$, we write $t^{\mathscr{A},v}$ for the value of t according to v. For the program statements we present a semantic description according to the axiomatic approach: For every program P and all postconditions N we define the weakest precondition M for which P is "totally correct with respect to the precondition M and the postcondition N", i.e.:

> Whenever M holds for an assignment $v \in \mathrm{STATE}$, then the program P terminates and N holds for (all) the assignment(s) (possibly) attained by executing P for v.

This use of weakest precondition is due to Dijkstra [3, 4]. His axioms and rules yield the following inductive definition of $w\wp$ for L_P-statements.

Definition 2.2. The function $w\wp: \mathrm{STAT} \times \mathscr{P}(\mathrm{STATE}) \to \mathscr{P}(\mathrm{STATE})$ is inductively defined by:

- $w\wp(\mathbf{skip}, N) := N$
- $w\wp(\mathbf{abort}, N) := \emptyset$
- $w\wp(x := t, N) := \{v \in \mathrm{STATE}: v\{t^{\mathscr{A},v}/x\} \in N\}$
  Here $v\{t^{\mathscr{A},v}/x\}$ is a notation for the mapping $w: \mathrm{PV} \to \bigcup_{s\in S} s^{\mathscr{A}}$ satisfying $w(x) = t^{\mathscr{A},v}$ and $w(y) = v(y)$ for all $y \neq x$
- $w\wp(P_1; P_2, N) := w\wp(P_1, w\wp(P_2, N))$
- $w\wp(\mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2\ \mathbf{fi}, N) := (\mathrm{Mod}(b) \cap w\wp(P_1, N)) \cup (\mathrm{Mod}(\neg b) \cap w\wp(P_2, N))$
- $w\wp(\mathbf{while}\ b\ \mathbf{do}\ P\ \mathbf{od}, N) := \bigcup_{k=0}^{\infty} M_k$, *where*

$$M_0 := \mathrm{Mod}(\neg b) \cap N$$

$$M_{k+1} := w\wp(\mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else\ skip\ fi}, M_k) = (\mathrm{Mod}(b) \cap w\wp(P, M_k)) \cup (\mathrm{Mod}(\neg b) \cap M_k)$$

Here $\mathrm{Mod}(b)$ and $\mathrm{Mod}(\neg b)$ are notations for the set of those assignments v, where $b^{\mathscr{A},v} = \mathbf{tt}$ or $b^{\mathscr{A},v} = \mathbf{ff}$, respectively.

This definition coincides with the denotational semantics presented, e.g., in [11]. For the nondeterministic variant we have to replace the definition of $w\wp$ for the

conditional statement by

$$\textbf{if } \bigsqcup_{i=1}^{n} (b_i \textbf{ then } P_i) \textbf{ fi}$$

$$:= \left(\bigcup_{i=1}^{n} \mathrm{Mod}(b_i) \right) \cap \bigcap_{i=1}^{n} ((\mathrm{Mod}(b_i) \cap w\wp(P_i, N)) \cup \mathrm{Mod}(\neg b_i)).$$

3. A finitary description of weakest preconditions

3.1. A many-sorted language of weak second-order logic

$\mathscr{L}_V$ is a many-sorted language of weak second-order logic. The programming language L_P is defined as a scheme language based on a signature Σ. Similarly $\mathscr{L}_V$ depends on Σ. In order to facilitate reasoning, additional functions and predicates may be introduced.

The sorts of $\mathscr{L}_V$ include the elements of S. For every $s \in S$, a sort sequ_s is added. The intended meaning of sequ_s is the set of finite sequences over s. As these sequences can be regarded as functions from the natural numbers into the interpretation of s, this language is called second order. The prefixed "weak" stems from the fact that only *finite* sequences are under discussion.

For every $s \in S$, there is an infinite set of first-order variables V_s^1, which extends the set of programming variables PV_s of sort s, and an infinite set of second-order variables V_s^2. Let $(C, F, \ldots; |\cdot|, \cdot(\cdot))$ be the set of $\mathscr{L}_V$-symbols, i.e.:

- the constant and function symbols of Σ are $\mathscr{L}_V$-symbols;
- apart from these arbitrary function and predicate symbols for the sorts in S may occur in $\mathscr{L}_V$;
- to handle the finite sequences, the function symbols $|\cdot|$ and $\cdot(\cdot)$ are contained in $\mathscr{L}_V$ (for any sort sequ_s).

$\mathscr{L}_V$-terms are defined as usual. Infix notation is used for basic operations or subscript notation s_i for $s(i)$ is used when comfortable and unambiguous. The formula of $\mathscr{L}_V$ are built from literals using $\wedge$, $\vee$, $\forall, \exists$ (first and second-order). We assume that programming variables do not occur bound in a formula: When quantification is used, we will rename first. **F** and **T** are special 0-ary predicate symbols. Negation and implication can be defined as operations on formulae.

The basis of $\mathscr{L}_V$'s semantics is an extension of the Σ-algebra $\mathscr{A}$ of Section 2.2. Let $\mathscr{A}^+$ be a structure for $\mathscr{L}_V$ and $\mathscr{A}^+|_\Sigma = \mathscr{A}$. Here $\mathbf{F}, \mathbf{T}, |\cdot|$ and $\cdot(\cdot)$ have to be interpreted in the obviously intended way, i.e. as truth values false, true and length and access function for sequences. STATE^+ denotes the set of assignments in $\mathscr{A}^+$. So STATE^+ consists of the mappings

$$v: \bigcup_{s \in S} V_s^1 \cup \bigcup_{s \in S} V_s^2 \to \bigcup_{s \in S} s^{\mathscr{A}} \cup \bigcup_{s \in S} (s^{\mathscr{A}})^*$$

which meet the conditions

$$v(x)\in s^{\mathscr{A}} \Leftrightarrow x\in V_s^1 \quad \text{and} \quad v(x)\in(s^{\mathscr{A}})^* \Leftrightarrow x\in V_s^2.$$

Here $(s^{\mathscr{A}})^*$ is the set of finite sequences over $s^{\mathscr{A}}$. The value of a term t^v (leaving out the subscript $\mathscr{A}^+$ as we would not consider any other structure) and the satisfaction relation $\mathscr{A}^+ \models \varphi[v]$ (keeping $\mathscr{A}^+$ to stress the dependency on $\mathscr{A}^+$) of a formula φ according to an assignment v are defined as usual. We write $\mathscr{A}^+ \models \varphi$ if $\mathscr{A}^+ \models \varphi[v]$ for every assignment v. For every formula φ we define the set of models:

$$\mathrm{Mod}(\varphi) := \{v\in \mathrm{STATE}^+ : \mathscr{A}^+ \models \varphi[v]\}.$$

As the set of $\mathscr{L}_V$-symbols and $\mathscr{A}^+$ are the extensions of Σ and $\mathscr{A}$, respectively, for each $s\in S$ the elements of EXP_s are $\mathscr{L}_V$-terms and $t^v = t^{\mathscr{A},v}$ for every $t\in\mathrm{EXP}_s$. We confined ourselves to total basic functions in order to get an embedding of the boolean terms. Obviously, for every $b\in\mathrm{EXP}_{\mathrm{bool}}$ there is a propositional formula B satisfying

$$b^v = \mathbf{tt} \Leftrightarrow \mathscr{A}^+ \models B[v] \quad \text{and} \quad b^v = \mathbf{ff} \Leftrightarrow \mathscr{A}^+ \not\models B[v].$$

We will assume such a fixed embedding and write b also for the formula related to b.

In Section 2.2 the mapping $w\wp : \mathrm{STAT}\times\mathscr{P}(\mathrm{STATE})\to\mathscr{P}(\mathrm{STATE})$ has been defined. Now we added further variables whose values are of no importance for and cannot be changed by the program. The extension of $w\wp$ to a mapping

$$\mathrm{STAT}\times\mathscr{P}(\mathrm{STATE}^+)\to\mathscr{P}(\mathrm{STATE}^+)$$

is then straightforward. We will write $w\wp$ for the latter again. $\mathscr{L}_V$ offers the opportunity to formulate important properties of programs and argue about them. To show this, we will present a formalization of $w\wp$ in $\mathscr{L}_V$.

3.2. *Expressing weakest preconditions: deterministic case*

We are now ready to present the definition of the formulae $\mathrm{wp}(P,\varphi)$.

Definition 3.1. Let $P\in\mathrm{STAT}$ *and* φ *be a formula of* $\mathscr{L}_V$. By recursion on P, we define a formula $\mathrm{wp}(P,\varphi)$ of $\mathscr{L}_V$ as follows:

- $\mathrm{wp}(\mathbf{skip},\varphi) :\equiv \varphi$
- $\mathrm{wp}(\mathbf{abort},\varphi) :\equiv \mathbf{F}$
- $\mathrm{wp}(x := t,\varphi) :\equiv \varphi\{t/x\}$ for $x\in\mathrm{PV}_s$ and $t\in\mathrm{EXP}_s$
- $\mathrm{wp}(P_1;P_2,\varphi) :\equiv \mathrm{wp}(P_1,\mathrm{wp}(P_2,\varphi))$
- $\mathrm{wp}(\mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2\ \mathbf{fi},\varphi) :\equiv (b\wedge\mathrm{wp}(P_1,\varphi))\vee(\neg b\wedge\mathrm{wp}(P_2,\varphi))$
- $\mathrm{wp}(\mathbf{while}\ b\ \mathbf{do}\ P\ \mathbf{od},\varphi) :\equiv \exists f\, E_{b,P,\varphi}(f,\bar{x})$, *where*

$$E_{b,P,\varphi}(f,\bar{x}) :\equiv (\neg b\wedge\varphi)\{f_{|f|-1}/\bar{x}\} \wedge\forall 0\leqslant i<|f|-1[b\wedge\mathrm{wp}(P,\bar{x}=f_{i+1})]\{f_i/\bar{x}\}\wedge f_0=\bar{x}$$

Here $\bar{x}$ is the list of the programming variables contained in the loop.

The meaning of the formula $E_{b,P,\varphi}(f,\bar{x})$ is as follows: f is a computation sequence consisting of storage states. The length $|f|$ of f corresponds to the number of repetitions of the while-loop (plus 1). The last component of f, i.e. $f_{|f|-1}$, gives the storage which results from the execution of the while-loop, f_0 the storage before entering the loop. The $\forall i$-part of the formula describes the transition of consecutive storage states inside the while-loop.

Usually wp is a rather complicated expression. It is hardly ever used explicitely, stronger conditions implying wp are used instead. In order to demonstrate our formalism, however, we consider the following example.

Example 3.2. Let P be the program:

$z := x;$
while $z \geqslant y$ **do** $z := z - y$ **od**

The program variables of P are x, y, z, which we suppose to be declared of sort integer. Let φ be an arbitrary formula having x, y, z as free variables. We compute $\mathrm{wp}(P,\varphi)$:

$$\mathrm{wp}(P,\varphi) \equiv \mathrm{wp}(\textbf{while } z \geqslant y \textbf{ do } z := z - y \textbf{ od}, \varphi)\{x/z\} \equiv \exists f\, E(f,y,x),$$

where

$$\begin{aligned} E(f,y,z) \equiv\ & f^2_{|f|-1} < f^1_{|f|-1} \wedge \varphi\{f^1_{|f|-1}/y, f^2_{|f|-1}/z\} \\ & \wedge \forall 0 \leqslant i < |f|-1[f^2_i \geqslant f^1_i \wedge (f^1_i, f^2_i - f^1_i) = (f^1_{i+1}, f^2_{i+1})] \\ & \wedge (f^1_0, f^2_0) = (y,z). \end{aligned}$$

We use f as a variable for sequences of pairs and f^1_i, f^2_i for the two components of the $(i+1)$th element of f. Obviously, these pairs satisfy $f_i = (y, z - i*y)$ for all $0 \leqslant i \leqslant |f|-1$, hence f is uniquely determined by y, z. So we obtain

$$\exists f\, E(f,y,z) \Leftrightarrow \exists n(z - n*y < y \wedge \forall i < n(z - i*y \geqslant y) \wedge \varphi\{z - n*y/z\}).$$

Therefore we get:

$$\mathrm{wp}(P,\varphi) \Leftrightarrow (y > 0 \wedge x \geqslant 0 \wedge \varphi\{x \bmod y/z\}) \vee (x < 0 \wedge x < y \wedge \varphi\{x/z\}).$$

In particular we found that our program P is totally correct with respect to the postcondition $\varphi \equiv (z = x \bmod y)$ if the precondition $x \geqslant 0 \wedge y > 0$ is satisfied – provided that $\mathrm{wp}(P,\varphi)$ formalizes $w\wp(P, \mathrm{Mod}(\varphi))$. This will be shown in Theorem 3.5.

We show now that the formula $\mathrm{wp}(P,\varphi)$ has all the properties we expect.

Theorem 3.3. (Normal form). *Let P be a program, φ a formula and $\bar{x}$ the list of the variables in P. Then*

(i) $\mathrm{wp}(P,\varphi) \Leftrightarrow \exists \bar{y}[\mathrm{wp}(P, \bar{x} = \bar{y}) \wedge \varphi\{\bar{y}/\bar{x}\}]$

(ii) $\forall \bar{y} \forall \bar{z}[\mathrm{wp}(P, \bar{x} = \bar{y}) \wedge \mathrm{wp}(P, \bar{x} = \bar{z}) \rightarrow \bar{y} = \bar{z}]$ *holds.*

Remark. The statements (i) and (ii) can be contracted to

$$\mathrm{wp}(P,\varphi) \Leftrightarrow \exists!\bar{y}[\mathrm{wp}(P,\bar{x}=\bar{y})\wedge\varphi\{\bar{y}/\bar{x}\}].$$

Proof. We use the notation $\varphi(\bar{x})$ to exhibit the occurrences of the variables $\bar{x}$ in φ and $\varphi(\bar{y})$ to abbreviate subsequent substitution.

(i) We use straightforward structural induction on P. As an example we present the case $Q;R$: By induction hypothesis for R we have

$$(1)\qquad \mathrm{wp}(R,\varphi) \Leftrightarrow \exists\bar{z}[\mathrm{wp}(R,\bar{x}=\bar{z})\wedge\varphi(\bar{z})].$$

The induction hypothesis for Q yields

$$(2)\qquad \mathrm{wp}(Q,\mathrm{wp}(R,\varphi)) \Leftrightarrow \exists\bar{y}[\mathrm{wp}(Q,\bar{x}=\bar{y})\wedge\mathrm{wp}(R,\varphi)\{\bar{y}/\bar{x}\}],$$
$$(3)\qquad \mathrm{wp}(Q,\mathrm{wp}(R,\bar{x}=\bar{z})) \Leftrightarrow \exists\bar{y}[\mathrm{wp}(Q,\bar{x}=\bar{y})\wedge\mathrm{wp}(R,\bar{x}=\bar{z})\{\bar{y}/\bar{x}\}].$$

Using (2) and (1) we deduce

$$\mathrm{wp}(Q;R,\varphi) \Leftrightarrow \exists\bar{z}[\exists\bar{y}(\mathrm{wp}(Q,\bar{x}=\bar{y})\wedge\mathrm{wp}(R,\bar{x}=\bar{z})\{\bar{y}/\bar{x}\})\wedge\varphi(\bar{z})].$$

By (3) we obtain

$$\mathrm{wp}(Q;R,\varphi) \Leftrightarrow \exists\bar{z}[\mathrm{wp}(Q,\mathrm{wp}(R,\bar{x}=\bar{z}))\wedge\varphi(\bar{z})] \equiv \exists\bar{z}[\mathrm{wp}(Q;R,\bar{x}=\bar{z})\wedge\varphi(\bar{z})].$$

The case of a loop is even easier, because here the definition already uses normal form expressions.

(ii) We proceed again by an easy structural induction on P. Consider for example a loop **while** b **do** Q **od**: Suppose we have

$$\neg b(f_m)\wedge f_m=\bar{y}\wedge\forall 0\leqslant i<m[b(f_i)\wedge\mathrm{wp}(Q,\bar{x}=f_{i+1})]\{f_i/\bar{x}\}\wedge f_0=\bar{x}$$

and

$$\neg b(g_n)\wedge g_n=\bar{z}\wedge\forall 0\leqslant i<n[b(g_i)\wedge\mathrm{wp}(Q,\bar{x}=g_{i+1})]\{g_i/\bar{x}\}\wedge g_0=\bar{x}.$$

Then trivially, $f_0=g_0$. By induction on j we further obtain:

$$f_j=g_j \quad \text{for } j=0,1,2,\ldots$$

So it follows that $m=n$ (if $m<n$ or $n<m$, we have $\neg b(f_m)$ and $b(g_m)$ or $\neg b(g_n)$ and $b(f_n)$, hence a contradiction, since $f_m=g_m$ or $f_n=g_n$, respectively). But then $\bar{y}=f_m=g_n=\bar{z}$. □

This theorem provides something like a "normal form representation" of $\mathrm{wp}(P,\varphi)$, which enables us to obtain further properties of wp easily. In the following theorem we show the monotonicity of wp and that wp behaves well on some of the logical operators.

Homomorphism theorem 3.4. wp *satisfies the following conditions*:

(i) (*Monotonicity of* wp): *If* $\mathscr{A}^+\models\varphi\rightarrow\psi$, *then also* $\mathscr{A}^+\models\mathrm{wp}(P,\varphi)\rightarrow\mathrm{wp}(P,\psi)$

(ii) $\mathrm{wp}(P,\mathbf{F}) \Leftrightarrow \mathbf{F}$

(iii) $\mathrm{wp}(P, \varphi \wedge \psi) \Leftrightarrow \mathrm{wp}(P, \varphi) \wedge \mathrm{wp}(P, \psi)$
(iv) $\mathrm{wp}(P, \varphi \vee \psi) \Leftrightarrow \mathrm{wp}(P, \varphi) \vee \mathrm{wp}(P, \psi)$
(v) $\mathrm{wp}(P, \exists x \varphi) \Leftrightarrow \exists x \mathrm{wp}(P, \varphi)$
(vi) $\mathrm{wp}(P, \forall x \varphi) \Leftrightarrow \forall x \mathrm{wp}(P, \varphi)$
(vii) $\mathrm{wp}(P, \varphi \to \psi) \Leftrightarrow (\mathrm{wp}(P, \mathbf{T}) \to \mathrm{wp}(P, \varphi)) \to \mathrm{wp}(P, \psi)$
(viii) $\mathrm{wp}(P, \neg \varphi) \Leftrightarrow \neg(\mathrm{wp}(P, \mathbf{T}) \to \mathrm{wp}(P, \varphi))$

Proof. (i), (ii), (iv) and (v) are immediate from 3.3(i).

(iii) We use both parts of Theorem 3.3. By (i) we have

$$\mathrm{wp}(P, \varphi(\bar{x})) \wedge \mathrm{wp}(P, \psi(\bar{x})) \Rightarrow \exists \bar{y}(\mathrm{wp}(P, \bar{x} = \bar{y}) \wedge \varphi(\bar{y}) \wedge \exists \bar{z}(\mathrm{wp}(P, \bar{x} = \bar{z}) \wedge \psi(\bar{z}));$$

by Theorem 3.3(ii) we have $\mathrm{wp}(P, \bar{x} = \bar{y}) \wedge \mathrm{wp}(P, \bar{x} = \bar{z}) \to \bar{y} = \bar{z}$, hence we obtain the desired result by using Theorem 3.3(i) for the postcondition $\varphi \wedge \psi$.

(vi) First we note that by Theorem 3.3(i)

$$\forall u \mathrm{wp}(P, \varphi(\bar{x}, u)) \Leftrightarrow \forall u \exists \bar{y}[\mathrm{wp}(P, \bar{x} = \bar{y}) \wedge \varphi(\bar{y}, u)],$$

so we have to show that the quantifiers on the right can be interchanged. By Theorem 3.3(ii),

$$\mathrm{wp}(P, \bar{x} = \bar{y}) \wedge \mathrm{wp}(P, \bar{x} = \bar{z}) \Rightarrow \bar{y} = \bar{z}.$$

Hence, we get

$$\forall u \exists \bar{y}[\mathrm{wp}(P, \bar{x} = \bar{y}) \wedge \varphi(\bar{y}, u)] \Rightarrow \exists \bar{y} \forall u[\mathrm{wp}(P, \bar{x} = \bar{y}) \wedge \varphi(\bar{y}, u)].$$

Bound variables were not allowed to occur in a program. Therefore, u does not occur in the formula $\mathrm{wp}(P, \bar{x} = \bar{y})$, and we obtain

$$\forall u \exists \bar{y}[\mathrm{wp}(P, \bar{x} = \bar{y}) \wedge \varphi(\bar{y}, u)] \Rightarrow \exists \bar{y}[\mathrm{wp}(P, \bar{x} = \bar{y}) \wedge \forall u \varphi(\bar{y}, u)].$$

Now by Theorem 3.3(i) the assertion follows.

(vii) $\Rightarrow$: From (i) and (iii) we have:

(1) $\mathrm{wp}(P, \varphi \to \psi) \wedge \mathrm{wp}(P, \varphi) \Rightarrow \mathrm{wp}(P, \psi)$,

(2) $\mathrm{wp}(P, \varphi \to \psi) \Rightarrow \mathrm{wp}(P, \mathbf{T})$.

From this we can deduce

$$\mathrm{wp}(P, \varphi \to \psi) \Rightarrow (\mathrm{wp}(P, \mathbf{T}) \to \mathrm{wp}(P, \varphi)) \to \mathrm{wp}(P, \psi)$$

$\Leftarrow$: First notice

$$(\mathrm{wp}(P, \mathbf{T}) \to \mathrm{wp}(P, \varphi)) \to \mathrm{wp}(P, \psi) \Leftrightarrow (\mathrm{wp}(P, \mathbf{T}) \wedge \neg \mathrm{wp}(P, \varphi)) \vee \mathrm{wp}(P, \psi).$$

As Theorem 3.3 yields

$$\mathrm{wp}(P, \mathbf{T}) \wedge \neg \mathrm{wp}(P, \varphi) \to \mathrm{wp}(P, \neg \varphi)$$

it remains to show

$$\mathrm{wp}(P, \neg \varphi) \vee \mathrm{wp}(P, \psi) \to \mathrm{wp}(P, \varphi \to \psi).$$

Also this is a consequence of (i).

(viii) Immediate consequence of (vii) since $\neg\varphi$ can be written as $\varphi \to \mathbf{F}$. □

So $\mathrm{wp}(P,\cdot)$ describes a monotonic homomorphism on weak second-order formulae. This fact can be restated as follows: For each formula φ define a new formula φ^* as follows:

$$\varphi^* :\equiv \mathrm{wp}(P,\varphi) \text{ if } \varphi \text{ is an atomic formula}$$

$$(\varphi \circ \psi)^* :\equiv \varphi^* \circ \psi^* \text{ for } \circ \in \{\wedge, \vee\}$$

$$(\varphi \to \psi)^* :\equiv (\mathrm{wp}(P,\mathbf{T}) \to \varphi^*) \to \psi^*$$

$$(Qx\varphi)^* :\equiv Qx\varphi^* \text{ for } Q \in \{\exists, \forall\}$$

Then $\mathrm{wp}(P,\varphi) \Leftrightarrow \varphi^*$.

We will show now that our formula $\mathrm{wp}(P,\varphi)$ exactly describes the semantically defined weakest precondition predicate $w\wp(P, \mathrm{Mod}(\varphi))$ for arbitrary postconditions that can be expressed by a formula of the language $\mathscr{L}_V$.

Theorem 3.5. *Let P be a program and φ a $\mathscr{L}_V$-formula. Then:*

$$\mathrm{Mod}(\mathrm{wp}(P,\varphi)) = w\wp(P, \mathrm{Mod}(\varphi)).$$

Proof. By structural induction on P: The only thing to show is that our formalization of the while loop behaves well. So let $P =$ **while** b **do** Q **od**, $\bar{x}$ a sequence containing all the variables in P, and

$$\begin{aligned} E^k_{b,Q,\varphi}(f,x) :\equiv\ & (\neg b \wedge \varphi)\{f_{|f|-1}/\bar{x}\} \\ & \wedge \forall 0 \leqslant i < |f| - 1[b \wedge \mathrm{wp}(Q, \bar{x} = f_{i+1})]\{f_i/\bar{x}\} \\ & \wedge f_0 = \bar{x} \\ & \wedge |f| \leqslant k+1 \end{aligned}$$

for all numbers $k \in \mathbb{N}$. Then wp(**while** b **do** Q **od**, φ) holds iff $\exists f E^k_{b,Q,\varphi}(f,\bar{x})$ holds for some $k \in \mathbb{N}$. Using

$$1 \leqslant |f| \leqslant k+1 \to |f| = 1 \vee \exists\bar{y}\exists g[1 \leqslant |g| \leqslant k \wedge f_0 = \bar{y} \wedge \forall 0 \leqslant i \leqslant |g|(g_i = f_{i+1})]$$

and logical rules we obtain

$$\exists f E^{k+1}_{b,Q,\varphi}(f,\bar{x}) \Leftrightarrow (\exists f E^k_{b,Q,\varphi}(f,\bar{x}) \wedge \neg b) \vee (b \wedge \exists\bar{y}(\mathrm{wp}(Q,\bar{x}=\bar{y}) \wedge \exists f E^k_{b,Q,\varphi}(f,\bar{y})))$$

Now we can apply the normal form theorem to obtain

$$(*) \qquad \exists f E^{k+1}_{b,Q,\varphi}(f,\bar{x}) \Leftrightarrow (\exists f E^k_{b,Q,\varphi}(f,\bar{x}) \wedge \neg b) \vee (b \wedge \mathrm{wp}(Q, \exists f E^k_{b,Q,\varphi}(f,\bar{x})))$$

Let

$$M_0 := \mathrm{Mod}(\neg b) \cap \mathrm{Mod}(\varphi)$$

$$M_{k+1} := (\mathrm{Mod}(b) \cap w\wp(Q, M_k)) \cup (\mathrm{Mod}(\neg b) \cap M_k)$$

for all $k \in \mathbb{N}$. Then $wp(\textbf{while } b \textbf{ do } Q \textbf{ od}, \mathrm{Mod}(\varphi)) = \bigcup_{k=0}^{\infty} M_k$ and it is easy to show that

$$\mathrm{Mod}(\exists f E^k_{b,Q,\varphi}(f, \bar{x})) = M_k \quad \text{for all } k \in \mathbb{N}$$

by induction on k. □

The following corollary builds the connection between our formalization of the predicate transformer wp and applications in the field of verification.

Corollary 3.6. *A program P is*

(i) *totally correct w.r.t. ψ and φ iff $\mathscr{A}^+ \models \psi \to \mathrm{wp}(P, \varphi)$,*

(ii) *partially correct w.r.t. ψ and φ iff $\mathscr{A}^+ \models \psi \to \neg \mathrm{wp}(P, \neg \varphi)$.*

As an example of an application we will present the proof for a rule concerned with total correctness. As pointed out before wp is hardly ever calculated explicitly. It is more convenient to use invariance properties (see [7, p. 144]). We have distinguished between the programming and the verification language. Therefore, we can assume the latter to contain a sort wf and a binary symbol $\prec$ interpreted as a well-founded ordering. Furthermore, we use F for embeddings of the other sets. So in our formalism the rule mentioned above takes the following form.

Theorem 3.7. *Consider a program* **while** b **do** P **od**. *Suppose a weak second-order formula ψ satisfies*

(1) $\psi \wedge b \to \mathrm{wp}(P, \psi)$

(2) $\psi \wedge b \to \mathrm{wp}(P, F(\bar{x}) \prec F(\bar{y}))\{\bar{x}/\bar{y}\}$, *where $\bar{y}$ are variables not occuring in P.*

Then $\psi \to \mathrm{wp}(\textbf{while } b \textbf{ do } P \textbf{ od}, \psi \wedge \neg b)$.

Proof. Use transfinite $\prec$-induction on the formula $\psi \to \mathrm{wp}(\textbf{while } b \textbf{ do } P \textbf{ od}, \psi \wedge \neg b)$ and the normal form theorem.

3.3. Nondeterminism

Let us finally have a short look on how to formalize the weakest precondition predicate in the case of nondeterministic programs. We consider finite nondeterminism expressed by Dijkstra's guarded commands [3, 4]:

$$\textbf{if } b_1 \textbf{ then } P_1 \;[]\; b_2 \textbf{ then } P_2 \;[]\; \ldots \;[]\; b_n \textbf{ then } P_n \textbf{ fi}$$

The operational behaviour of a program of this kind can be described by: One of the subprograms P_i for which the condition b_i holds is carried out, but it is not determined which one if more than one of the b_i's happen to be true. If every condition yields false, the execution results in an error. Consequently, the computation process can no longer be represented by a sequence of storage states. In fact, a finite tree, whose branches represent all possible computations, has to be used. Therefore, the

formalization of the weakest precondition has to be modified:

- The verification language is supposed to contain new sorts: finite sets and finite labelled trees with labels in s ($s \in S$). The nodes themselves can be regarded as natural numbers. Then our access function $\cdot(\cdot)$ takes a tree τ and a natural number i as arguments and gives a result in s if i is a node of τ: We fix 0 as the root and let $n(\tau)$, $l(\tau)$, $\mathrm{succ}(\tau, i)$ denote the set of nodes of τ, leaves of τ and successor nodes of i in τ (if $i \in n(\tau)$) respectively. We will use a predicate symbol $\in$ for the membership relation and a function symbol h for the height of a tree.
- The definition of $\mathrm{wp}(P, \varphi)$, where P is a program in the extended programming language $\mathrm{STAT}^{\mathrm{nondet}}$ and φ a formula of the modified verification language, is as follows:
  - $\mathrm{wp}(\textbf{if } \square_{i=1}^{n} (b_i \textbf{ then } P_i) \textbf{ fi}, \varphi) :\equiv \bigvee_{i=1}^{n} b_i \wedge \bigwedge_{i=1}^{n} (b_i \to \mathrm{wp}(P_i, \varphi))$
  - $\mathrm{wp}(\textbf{while } b \textbf{ do } P \textbf{ od}, \varphi) :\equiv \exists \tau F_{b,P,\varphi}(\tau, \bar{x})$, where

$$F_{b,P,\varphi}(\tau, \bar{x}) :\equiv \forall i \in l(\tau)(\neg b \wedge \varphi)\{\tau_i/\bar{x}\}$$

$$\forall i \in n(\tau)[i \notin l(\tau) \to (b \wedge \mathrm{wp}(P, \exists j \in \mathrm{succ}(\tau, i)(\bar{x} = \tau_j)))\{\tau_i/\bar{x}\}]$$

$$\wedge \tau_0 = \bar{x}.$$

  - For all other programming language constructs the definition of wp remains unchanged.

Now the analogies of the above theorems can be proved. Since these proofs can be carried out using the same techniques as in the deterministic case, we omit them and simply state the theorems.

Theorem 3.8 (Normal form). *Let P be a program, φ a formula and $\bar{x}$ the list of the variables in P. Then*

(i) $\mathrm{wp}(P, \varphi) \Leftrightarrow \exists \tau[h(\tau) = 1 \wedge \mathrm{wp}(P, \exists i \in l(\tau)(\bar{x} = \tau_i)) \wedge \forall i \in l(\tau)\varphi\{\tau_i/\bar{x}\}]$

(ii) *The set of trees τ which satisfy $\mathrm{wp}(P, \exists i \in l(\tau)(\bar{x} = \tau_i))$ is either empty or contains a least element (w.r.t. the subtree ordering).*

In a less formal way, (i) can be restated as

$$\mathrm{wp}(P, \varphi) \Leftrightarrow \exists \{\bar{y}_1, \ldots, \bar{y}_m\} [\mathrm{wp}(P, \bar{x} \in \{\bar{y}_1, \ldots, \bar{y}_m\}) \wedge \forall 1 \leqslant i \leqslant m \varphi(\bar{y}_i)]$$

and, by (ii), if m is taken minimal, then $\{\bar{y}_1, \ldots, \bar{y}_m\}$ is unique.

Theorem 3.9 (Properties of wp).

(i) *If $(\varphi \to \psi)$ holds in the standard interpretation for every assignment, then so does $(\mathrm{wp}(P, \varphi) \to \mathrm{wp}(P, \psi))$*

(ii) $\mathrm{wp}(P, \mathbf{F}) \Leftrightarrow \mathbf{F}$

(iii) $\mathrm{wp}(P, \varphi \wedge \psi) \Leftrightarrow \mathrm{wp}(P, \varphi) \wedge \mathrm{wp}(P, \psi)$

(iv) $\mathrm{wp}(P, \forall u \varphi) \Leftrightarrow \forall u \mathrm{wp}(P, \varphi)$

Compatibility of disjunction and existential quantification are in general not true for nondeterministic programs. Clearly, the easy implications

$$\mathrm{wp}(P,\varphi) \vee \mathrm{wp}(P,\psi) \Rightarrow \mathrm{wp}(P,\varphi \vee \psi)$$

and

$$\exists u \mathrm{wp}(P,\varphi) \Rightarrow \mathrm{wp}(P,\exists u \varphi)$$

remain true, as they follow from monotonicity. In a sense the reverse implications express determinacy. If $\mathrm{wp}(P,\cdot)$ commutes with $\exists$, we have

$$\mathrm{wp}(P,\mathbf{T}) \Rightarrow \exists \bar{y} \mathrm{wp}(P,\bar{x}=\bar{y}).$$

The new normal form theorem is not strong enough to maintain the implication and negation property. Intuitively it is clear that, even if termination is ensured, $\mathrm{wp}(P,\neg\varphi)$ and $\neg\,\mathrm{wp}(P,\varphi)$ need not coincide: The former tells us that $\neg\varphi$ holds after every possible execution of P, while the latter only expresses the existence of a possible execution of P whose result satisfy $\neg\varphi$.

Can we add other constructs like a nondeterministic assignment statement? In [1] it had been pointed out before that even $\mathscr{L}_{\omega_1\omega}$ does not allow formalization of $w\wp$ in this case. At first sight this may be surprising because the assignment statement itself can be expressed easily by

$$\mathrm{wp}(x := y \cdot \psi,\varphi) :\equiv \exists y \psi \wedge \forall y(\psi \rightarrow \varphi\{y/x\})$$

The problem is the formalization of $w\wp$ for the loop. It makes use of the fact that there are only finitely many possibilities. As soon as this is guaranteed we can proceed as before. For a more detailed discussion of this see [1].

Acknowledgements

We thank the referees of the earlier version of this paper for their helpful comments.

References

[1] R.J.R. Back, Proving total correctness of nondeterministic programs in infinitary logic, *Acta Inform.* **15** (1981) 233–249.

[2] R.J.R. Back, Predicate transformers and higher order logic, in: J.W. de Bakker, W.-P. de Roever, G. Rozenberg, eds. *Proc. of the REX Workshop, Beekbergen 1992*, Lecture Notes in Computer Science, Vol. 666 (Springer, New York, 1993).

[3] E.W. Dijkstra, Guarded commands, nondeterminacy and formal derivation of programs. *Comm. ACM* **18** (1975) 453–457.

[4] E.W. Dijkstra, *A Discipline of Programming* (Prentice-Hall, Englewood Cliffs, NJ, 1976).

[5] E.W. Dijkstra and C.S. Scholten, *Predicate Calculus and Program Semantics*, Texts and Monographs in Computer Science (Springer, New York, 1990).

[6] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specifications* 1. *Equations and Initial Semantics*, EATCS Monographs in Theoretical Computer Science, Vol. 6 (Springer, Berlin, 1985).

[7] D. Gries, *The Science of Programming*, Texts and Monographs in Computer Science (Springer, New York, 1981).
[8] C.A.R. Hoare, An axiomatic basis for computer programming, *Comm. ACM* **12** (1969) 576–583.
[9] C.R. Karp, *Languages with Expressions of Infinite Length* (North-Holland, Amsterdam, 1964).
[10] H.J. Keisler, *Model Theory of Infinitary Logic* (North-Holland, Amsterdam, 1971).
[11] J. Loeckx and K. Sieber, *The Foundations of Program Verification* (Teubner, Stuttgart, 1984).
[12] R.A. Platek, Making computers safe for the world: An introduction to proofs of programs. Part I, in: *Logic and Computer Science*, Lecture Notes in Mathematics, Vol. 1429 (Springer, Berlin, 1990).